12

Probando el Control acceso

Transcripción

[00:00] Bien, vamos a volver a ver este gráfico para entender un poco qué es lo que está sucediendo. Recuerden acerca de los filtros, los filtros de Java son llamados y después los de Spring subsecuentemente y entran al dispatcherServlet de Spring y los filtros que implementamos ahí hacen el filtrado correspondiente valga la redundancia, para mandarlo al Controller. ¿Qué es lo que está pasando aquí?

[00:26] Como podemos ver, por más que yo estoy enviando requests con token válido, yo estoy obtenido un error forbidden, pero mi filtro ni siquiera está haciendo llamado. ¿Esto qué quiere decir? El filtro de Spring en esta instancia porque recuerden que mi filtro, el filtro que hemos implementado extiende de OncePerRequestFilter, que es propio de Spring.

[00:52] Entonces por lo tanto necesito decirle a Spring: “Sprint, antes de llamar a tu filtro, llama a mi filtro”. Porque lo que está sucediendo es que como yo ya configuré aquí, este filtro está siendo llamado antes que el que yo he implementado.

[01:10] Y por lo tanto está bloqueando todos mis requests, porque él solamente valida: “Okay, este método está abierto”. Listo, está abierto, pero todos los demás están cerrados, porque no tiene cómo validar una logia, que yo estoy usando aquí en mi propio filtro que he creado.

[01:26] ¿Qué es lo que tengo que hacer para eso? Tengo que encontrar una forma de decirle que agregue mi filtro primero antes que el filtro de Spring p

defecto. ¿Spring es tan personalizable que me permitirá hacer eso? Pues claro, obvio, y lo vamos a hacer ahora para permitir estos requests. ¿Cómo?

[01:49] Con un ejemplo aquí después de anyRequest, lo que yo le voy a decir es que me agregue un filter, aquí no porque estamos ante authenticated. Al final. Y lo que yo le voy a decir es qué también, .and(), .addFilterBefore. ¿Qué filtro quiero agregar antes? Quiero agregar mi SecurityFilter, pero yo necesito una instancia de mi SecurityFilter.

[02:28] Entonces lo que voy a hacer es, como ustedes ya saben, private, llamo a mi SecurityFilter. Teniendo en cuenta que este med.voll.api mi SecurityFilter. Como es un componente lo voy a inyectar, le voy a decir @Autowired, y esta instancia yo se la voy a meter aquí.

[02:50] Y listo, entonces aquí mi .addFilterBefore(securityFilter) y también un tipo de autenticación, que es UsernamePasswordAuthenticationFilter.class. Y esto acá les puede sonar un poco extraño al inicio. Esto es porque este tipo de filtro UsernamePasswordAuthenticationFilter lo que va a hacer es validar que en efecto el usuario que está iniciando la sesión existe y que ya está autenticado.

[03:20] Esto posiblemente nos lance otro error, por ejemplo aquí ya en compilación ya vemos que no necesitamos otro and, entonces lo borramos y lo dejamos en build. Vamos a guardar. Esperamos un momento. Como pueden ver, aquí lo que yo hice fue decirle explícitamente “agrega un filtro antes aquí, porque yo voy a implementar UsernamePasswordAuthenticationFilter en mi SecurityFilter.”

[03:52] Venimos aquí, enviamos, vemos que sigue prohibido y login sigue funcionando. ¿Entonces qué es lo que tengo que hacer ahora? Como estoy usando UsernamePasswordAuthenticationFilter, este método de aquí, esta clase, lo que hace es validar que mi usuario en sí tiene una sesión iniciada en mi sistema, en mi API, pero recuerden que tenemos un tipo de sesión stateless.

[04:21] ¿Cómo tú puedes forzar un login de usuario por cada request? Exacto, lo que vamos a hacer aquí en el filtro ahora es que si el token no es nulo, reemplazamos por "Bearer", obtenemos el subject del token. Entonces aquí vamos a hacer un pequeño refactor, porque lo que yo le voy a decir, es que la variable subject va a ser igual a TokenService.getSubject, yo voy a borrar estas dos líneas porque ya no las necesitamos.

[04:56] Vamos a mantener nuestro código limpio, borramos esta de aquí y borro esto de aquí. Y si subject no llega nulo if subject no es nulo, eso ya me dice que es válido. Si subject no llega a nulo es que el token ya está válido. Token valido. Perfecto. ¿Qué es lo que sigue?

[05:26] Si es token válido, por lo tanto ahora yo lo que quiero hacer es forzar un inicio de sesión en mi sistema. ¿Esto cómo lo hago? Voy a venir aquí y le voy a decir private UserRepository. No es este de aquí. Es UsuarioRepository, como ustedes ya deben saber @Autowired y aquí lo que yo le voy a decir es que usuario = usuarioRepository.FindByLogin y aquí lo que yo le voy a mandar es el subject.

[06:19] Entonces con esto él ya debería ser capaz de retornar mi usuario por login y hacer el llamado al método para que inicie sesión en este momento. Vamos a ver. Lo que tengo que hacer ahora primero es invocar a una clase de Spring SecurityContextHolder y aquí lo que voy decir es getContext.

[06:49] Del contexto de seguridad que tiene en este momento Spring, yo le quiero decir setAuthentication, y este Authentication va a ser mi usuario. De la misma forma en la que lo hicimos anteriormente le voy a mandar mi usuario como mi objeto de autenticación que yo deseo usar. Pero vemos que el usuario no está compilando. ¿Por qué?

[07:12] Porque yo necesito transformar este usuario en un objeto de authentication entonces lo que yo hago ahora es decirle que authentication es una variable que yo voy a crear aquí y aquí es donde va a suceder toda la magia,

porque authentication no es un var. Venimos aquí var authentication = usernamePasswordAuthenticationToken.

[07:42] Esto va a recibir, ustedes ya conocen esto. Esto es un new. Ustedes ya han visto esto. Es el usuario. ¿Qué más va a recibir? No va a tener las credenciales porque no están generadas y aquí también va a tener el usuario., porque yo ya tengo el objeto en la base de datos usuario.getAuthorities().

[08:06] Voy a darle un salto de línea aquí para que quede más legible, punto y coma al final y con esto yo ya le estoy diciendo a Spring mi siguiente lógica. Le estoy diciendo que si el token no llega nulo, por ejemplo, primero que todo yo quiero el header authorization si no es nulo, entonces voy a hacer un replace, para que quede un poco más legible lo que yo voy a poner aquí.

[08:30] Voy a reemplazar esta variable diciéndole que este va a ser el authHeader. Si el authHeader no es nulo, entonces token y voy a declarar aquí var token. Entonces si no llega nulo, reemplazamos el authHeader y eso sería el token, el authorization token. Obtenemos el subject y preguntamos nuevamente si el subject no es nulo, el token está válido y con esto encontramos al usuario.

[09:08] Con estas dos líneas le decimos a Spring: “este login es válido para mí” porque yo estoy autenticando este usuario ahora con mi request y yo te estoy verificando que el usuario existe. Entonces yo estoy acá forzando, forzamos inicio de sesión.

[09:29] Ahora, que forzamos ese inicio de sesión lo que hacemos es, en el SecurityContext de Spring, vamos a setear manualmente esa autenticación, de modo que sí venimos aquí, miren, a mi configuración ya para los demás requests el usuario ya va a estar autenticado.

[09:51] Okay vamos a probar ahora nuestro código. Vamos a nuestro listado de médicos, enviamos. Y miren cómo está ahora. Ya está recibiendo los datos porque está forzando una autenticación de usuario.

[10:06] En el delete, por ejemplo, sigue prohibido. En el get también sigue prohibido porque no estamos enviando ningún token. En mi login, voy a enviar también sin ningún tipo de cabecera y también vemos que está funcionando.

[10:20] Entonces recordamos los casos de uso uno a uno. Login abierto sin ningún dibujo de autenticación: funciona. Endpoint sin los datos de autenticación, sin el header de autenticación me da prohibido, funciona bien. Y cuando yo estoy enviando el token correcto me está mandando lo que yo necesito.

[10:42] Entonces esta forma ya hemos implementado ya correctamente y cubriendo la mayoría de casos de uso cómo hacer un proceso de autorización en Spring haciendo uso de los métodos de Spring. Vamos a hacer un pequeño recap de lo que hemos visto ahora, porque quizás es un poco confuso.

[11:01] El error inicial era que este filtro no estaba haciendo llamado por el otro filtro Spring porque configuramos en SecurityConfigurations que cada request tenía que ser autenticado aún no teníamos esto, entonces el filtro de Spring por defecto venía antes que el nuestro y al filtro de Spring no le habíamos dicho que tenía que llamar al nuestro.

[11:28] Recuerden este diseño siempre. Un filtro tiene que llamar al siguiente, si no nunca es llamado. Nuestro filtro no era llamado, por lo tanto lo que tuvimos que hacer es decirle: agrega un filtro antes de este filtro usando este tipo de autenticación: usernamePasswordAuthenticationFilter.

[11:47] ¿Por qué? Porque con este tipo de filtro usando, como nosotros obtenemos el token y con el token extraemos el usuario, el nombre del usuario aquí, que sería el subject, vamos a ver aquí, extract username.

[12:07] Extraemos el name, incluso hasta podemos hacerlo nombreUsuario para hacerlo un poco más autoentendible nuestro código, siempre es bueno. Entonces lo que hacemos aquí es que si el nombre de usuario no llega a null significa que para mí entonces este token es válido. Ya está aquí.

[12:28] Si esto no es nulo entonces para mí, para mi regla de negocio esto ya es válido. ¿Qué hago ahora? Fuerzo una autenticación de este usuario, username authentication token con el usuario que yo acabo de tener de la base de datos y con eso, yo ya le digo al context: “Spring por si acaso este usuario ya está autenticado”.

[12:49] Su autenticación es válida y te lo estoy indicando aquí. Por lo tanto en mi SecurityConfigurations ya esta condición en .anyRequest .authenticated ya se vuelve true, porque ya para Spring ese usuario ya fue autenticado. Entonces como siempre les digo estudien mucho, pónganle mucho empeño.

[13:10] Yo sé que toda esta parte de seguridad puede ser un poco confusa, pero no se preocupen, con un poco de práctica todo se vuelve más simple. Nos vemos.